Out of Core FFTs in a Parallel Application Environment

Christopher Lee Kuszmaul[1]

Report RND-93-013 September 1993

NAS Systems Development Branch
NAS Systems Division
NASA Ames Research Center
Mail Stop 258-6
Moffett Field, CA 94035-1000

[1] Computer Sciences Corporation, NASA Ames Research Center, Moffett Field, CA 94035-1000

Out of Core FFTs in a Parallel Application Environment

Christopher Lee Kuszmaul

August 6, 1993

Abstract

A principle mission at the NAS facility is to establish highly parallel computer systems supporting full scale production use by 1996. In order to fulfill this objective, parallel systems must support high speed scalable I/O — suitable for handling output of large scale numerical aerodynamic simulation.

Pursuant to this goal, we seek to execute an 'out of core' radix 2 Fast Fourier Transform (FFT) as rapidly as possible. By 'out of core' we mean that the size of the problem to be solved is too large to fit in normal memory.

We implement out of core methods on each of two computer architectures: the CM5 with a scalable disk array (SDA), and the Intel iPSC/860 with a Concurrent File System (CFS).

In the case of the out of core FFT, the most successful I/O approach known is to apply an intermediate transpose of the data when viewed as a two dimensional matrix [2]. We implement and evaluate three I/O methods for performing the required transpose. The first method (row/col) transposes by exchanging rows and columns of the data set, the second (diagonal) exchanges diagonals, and the third (recursive) applies a recursive divide and conquer approach.

Only the recursive method can be feasibly implemented on the CM5 SDA because it does not support 'independent access' — the ability for each processor to maintain an independent pointer to a given file on disk.

We discover that in absolute terms, the CM5 SDA, using the recursive method, outperforms the iPSC/860 CFS using any of the three methods in solving out of core FFTs. However, the recursive method shows a rapid decline in performance with problem size. The iPSC/860 CFS, using the diagonal method or the row/col method scales well with problem size.

We conclude that if the CM5 SDA had the independent access provided by the iPSC/860 CFS, then the CM5 SDA would scale as well as the iPSC/860 CFS, but at a much higher level of absolute performance.

1 Introduction

We seek to execute an 'out of core' radix 2 Fast Fourier Transform (FFT) as rapidly as possible. By 'out of core' we mean that the size of the problem to be solved is too large to fit in normal memory.

Some out of core applications have proven successful using current I/O technology. For example, [10] has shown that dense matrix solvers run efficiently out of core on the iPSC/860 using the Concurrent File System (CFS). However, such applications do not reflect the relatively low data reuse that characterize algorithms that solve NAS problems [10].

The FFT reflects the data reuse inherent in many NAS applications, such as rapid solution of certain partial differential equations for fluid flow [3]. The FFT also has many other applications such as in signal and image processing, and polynomial manipulation [4].

Rapid solution of out of core FFTs requires a transpose operation upon the data when viewed as a two dimensional matrix [2]. We employ three alternative transpose methods. One is a naive demand paging implementation that exchanges the rows and columns of the matrix. The other two are optimal for some model of an I/O system. A method that exchanges the diagonals of the matrix is optimal under the Vitter/Shriver I/O model [12]. A method that applies a recursive divide and conquer approach is optimal under the Vitter/Schriver model when there is no 'independent' access. 'Independent' access is the ability for each processor to maintain an independent pointer to a given file on disk [12].

We implement the out of core FFTs on each of two parallel systems, the CM5 Scalable Disk Array (SDA) and the iPSC/860 Concurrent File System (CFS).

The rest of this paper is organized into a section (2) on our approach, describing the algorithms and architectures in more detail. Section 3 summarizes our results, describing performance for each architecture for each out of core algorithm. Section 3 also shows the in core performance for FFTs for

both architectures to establish the difference in performance and scalability for in core vs. out of core solutions on these systems. Finally, in sections 4 and 5 we draw our conclusions and identify future work.

2 Approach

We compare the performance on the CM5 and the Intel iPSC/860 using three different I/O approaches for the FFT. The first is optimal, given independent read head access as provided by the Intel CFS. The second is optimal given non-independent read head access as provided by the CM5 SDA. Finally, as a basis for comparison the last algorithm is a naive demand paging algorithm for the transpose. In the rest of this section the overall FFT implementation is described, followed by descriptions of each of the three underlying I/O algorithms. First, we briefly describe the CM5 SDA and Intel CFS I/O systems.

2.1 The I/O Systems

We employ two different parallel I/O systems. One is the CM5 Scalable Disk Array (SDA) , the other is the Intel Concurrent File System (CFS).

2.1.1 CM5 SDA

The scalable disk array is a collection of sets of high speed commodity disks directly connected to the CM5 interprocessor communication networks [11]. The SDA has a peak rated performance of 264 MB/sec and a peak storage capacity of 200 Gbytes. File access is constrained such that all the processors use the same file pointer. Therefore it is not possible to implement the diagonal or row column I/O algorithms in parallel, as they require multiple pointers (independent access) into a file. The SDA must use the theoretically inferior recursive algorithm. The system at NAS consists of 128 compute nodes, and 32 active disks, with 37 Gbytes of total storage and a peak I/O rate of 32 MB/sec.

All code for the CM5 algorithms was written in CM-Fortran. Compilation was performed using the

```
-vu -cm5 -extend_source
```

options to the CMFortran compiler, cmf, version:

```
[CM5 VecUnit 2.1 Beta 0.1].
```

In core FFTs were performed via the cmssl library *detailed-fft* call [7].

2.1.2 Intel CFS

The Concurrent File System is the disk I/O subsystem of the Intel iPSC/860, a hypercube based MIMD parallel computer. The CFS at the NAS facility consists of 10 I/O nodes and 10 SCSI disks, with a theoretical peak performance of 10MB/sec, and 7.6 GB of storage [8]. Independent read/write access is supported by CFS.

All code for the iPSC/860 algorithms was written in Intel Fortran. Compilation was performed using the default options to if77, version:

```
[if77/NX SGI Rel 4.0].
```

In core FFTs were performed via the intel math library *cfft* call [7] in conjunction with modifications recommended in [3].

2.2 The FFT algorithms

The practical objective is to execute a very large radix 2 FFT as rapidly as possible. Alternative radix algorithms exhibit similar data movement patterns and overall complexity [4]. The most obvious implementation choice is simply to access disk memory at the time that the corresponding data is required for further computing. This corresponds to a demand paging virtual memory [9].

One implementation choice for hierarchical memories is known as the four step FFT [2]. This approach calls for demand paging for the first half of the computation, then a small amount of added floating point computation, followed by an intermediate reorganization of the data that sustains the level of data reuse (locality) present in the first part of the algorithm, and finally demand paging using the new, superior data layout. The intermediate reorganization is a transpose of the data when the data is viewed as a 2 dimensional matrix.

2.3 The Transpose Algorithms

The objective of the transpose algorithm is to rearrange the data on disk so that in the later stages of the FFT the data can be accessed as efficiently as possible. The FFT accesses data first in rows, then later in columns. If the rows are striped across the disks, then the columns will tend to be concentrated one per disk (or a column on a few disks — depending on choice of block size, number of disks, and the problem size). So the FFT first accesses data efficiently as rows, then inefficiently as columns. This inefficiency arises because when a column is read, only the disks on which it resides can be used for I/O bandwidth - the I/O bandwidth provided by parallel disks is lost. It is at the point of transferring between accessing by rows to accessing by columns that the transpose can be used, since the transpose exchanges rows and columns.

We evaluate three transpose algorithms, a 'naive' algorithm, an optimal algorithm for the case of independent access, and an optimal algorithm in the case of non independent read head access. The optimality of each of these algorithms is based on the Vitter/Shriver model of parallel I/O [12].

Any transposition algorithm serves to exchange block (i,j) with block (j,i) on disk. The objective of a given transpose algorithm is to maximize the parallelism in each step of the transpose, under the constraints of the underlying I/O architecture.

One obvious choice is simply to read in each row (or column) and write it out as a column (or row). This approach risks a load imbalance: a row striped across disks will transpose into a column concentrated on a single disk. We implement this 'naive' algorithm and refer to it as the row/col algorithm.

EXAMPLE 1: Row/Col Algorithm on 4 Disks with 4x4 array:

Start:

DISK0	DISK1	DISK2	DISK3
A*	B*	C*	D*
E	F	G	H
I	J	K	L
M	N	O	P

Goal:

DISK0	DISK1	DISK2	DISK3
A*	E	I	M
B*	F	J	N
C*	G	K	O
D*	H	L	P

The first starred (*), row (A B C D) is striped across the 4 disks. The row/col algorithm reads in this row efficiently, since all disks are used to read, but then the entire row must be written to DISK0, serializing the algorithm.

An algorithm that is theoretically optimal has been derived [12]. An equivalent algorithm, for problem and block sizes of interest, serves to access blocks along a diagonal of the matrix. Each element on a diagonal is neither in the same processor nor in the same memory location. To achieve the theoretical optimality, any given row of blocks must be striped across all the disks. This algorithm will be referred to as the *diagonal* method.

EXAMPLE 2: Diagonal Algorithm on 4 Disks with 4x4 array:

Start:

DISK0	DISK1	DISK2	DISK3
A	B*	C	D
E	F	G*	H
I	J	K	L*
M*	N	O	P

Goal:

DISK0	DISK1	DISK2	DISK3
A	E	I	M*
B*	F	J	N
C	G*	K	O
D	H	L*	P

Every diagonal, such as the starred (B G L M)
is striped across the 4 disks. The diagonal algorithm
reads in efficiently, since all disks are used to read.
Since the goal position for the diagonal elements is
also striped across all disks, the diagonal algorithm
also writes out efficiently.

The diagonal method algorithm requires independent access. The CM5 SDA does not provide this capability. Therefore, a recursively defined transpose algorithm is applied. The algorithm is defined by dividing the matrix into its 4 subsquares, exchanging the upper right and lower left hand squares, and then applying the algorithm recursively on each subsquare. This follows, since the transpose of the subsquares concatenated with the transpose of the interior of the subsquares is the transpose of the matrix:

$$\begin{pmatrix} X^T Y^T \\ Z^T W^T \end{pmatrix} = \begin{pmatrix} XZ \\ YW \end{pmatrix}^T$$

Some portion of a row striped across all disks must be read or written during each I/O transaction, and at the initial stage of the algorithm only one element of a given row can be placed into the correct column without disk access conflict. In each stage of the algorithm the number of elements that can be placed into the proper column doubles. (This corresponds to the optimal transpose in [1], where M/B is 2.)

EXAMPLE 3: Recursive Algorithm on 4 Disks with 4x4 array:

Start:

DISK0	DISK1	DISK2	DISK3
A	B	C*	D*
E	F	G*	H*
I	J	K	L
M	N	O	P

INTERMEDIATE Goal:

DISK0	DISK1	DISK2	DISK3
A	E	I	M
B	F	J	N
C*	G*	K	O
D*	H*	L	P

Every subblock, such as the starred (C D G L), is striped across half the disks. The recursive algorithm reads in with 50 percent efficiency, since half the disks are used to read. Since the intermediate goal position for each subblock is also striped across half the disks, the recursive algorithm writes at the same efficiency as it reads.

This algorithm has the advantage that it can be used without independent read head access, however, the algorithm must repeat each subblock movement recursively within each subblock, so a factor of log(N) more steps is required, where N is the number of disks.

3 Results

Here we show the performance of large out of core FFTs on the CM5 and the iPSC/860. We also include a section on the in core performance of the FFT on the two machines as a basis for comparing the scalability and performance of the in core with the out of core capabilities of each system.

3.1 Out of Core FFT Performance

The newer generation CM5 outperforms the iPSC/860 in actual performance time for out of core FFTs by a factor of five. Below we summarize the performance results for each of three transpose methods on each architecture.

3.1.1 Row/Col Method

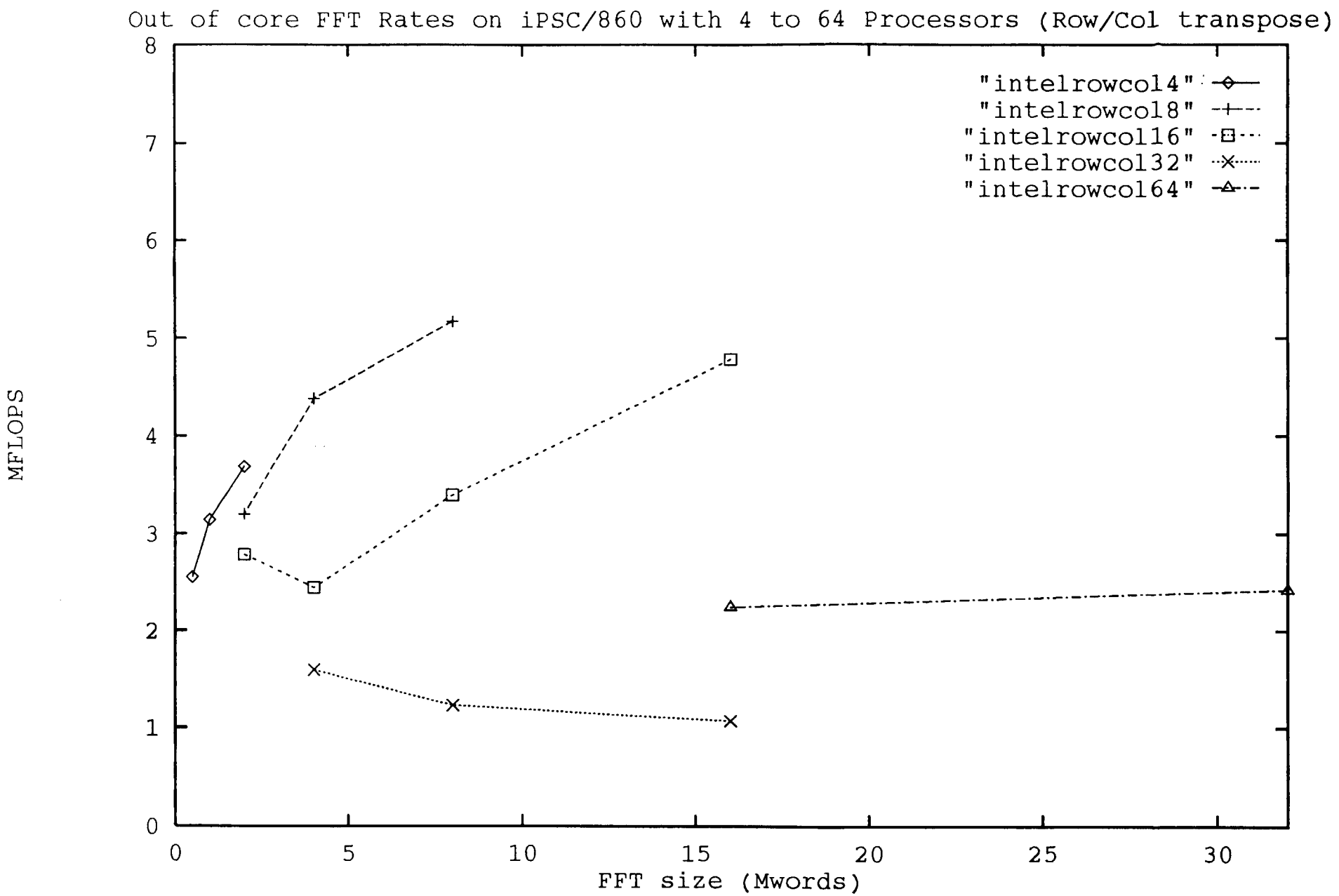
The Row/Col method is theoretically pessimal in the sense that it is expected to require serialization of data access to an I/O system. On the CM5 this is so because there is no software mechanism that allows a user program to access a column of data stored in row major order (this corresponds to the fact that independent mode is dissallowed on SDA requests) [11]. The software on the iPSC/860 allows independent access. This does not guarantee the underlying system will avoid serialization. The CFS does not allocate blocks in a regular fashion to disks, but instead uses a 'free list' [8]. This at least partly randomizes the block placement on the disks. As such, the pessimal performance does not occur on CFS, but neither does the optimal performance.

Graph 1 shows the performance of the iPSC/860 in solving out of core FFTs for sizes varying from .25 Mwords to as many as 32 Mwords. (A complex word occupies 16 bytes and corresponds to one point of data) The MFLOP number is derived from the total run time of the out of core solver divided into the total number of floating point operations required for the

given problem size. In particular, an FFT of N complex words requires $5 * N * \log N$ real floating point operations.

We see that the performance using the row/col method does not correlate well either to the number of processors nor the size of the problem. We see that for each choice of number of processors, the algorithm scales well - as problem size increases, performance does not degrade. Overall, it appears to be difficult to predict the precise performance of the iPSC/860 for a given number of processors and a given problem size. In general, one can expect to achieve a maximum of 5 MFLOPS and a minimum of 1 MFLOP for this out of core algorithm on the iPSC/860, independent of the problem size or the number of processors.

Graph 1


Out of core FFT Rates on iPSC/860 with 4 to 64 Processors (Row/Col transpose)
MFLOPS
FFT size (Mwords)
"intelrowcol4"
"intelrowcol8"
"intelrowcol16"
"intelrowcol32"
"intelrowcol64"
0
1
2
3
4
5
6
7
8
0
5
10
15
20
25
30

3.1.2 Diagonal Method

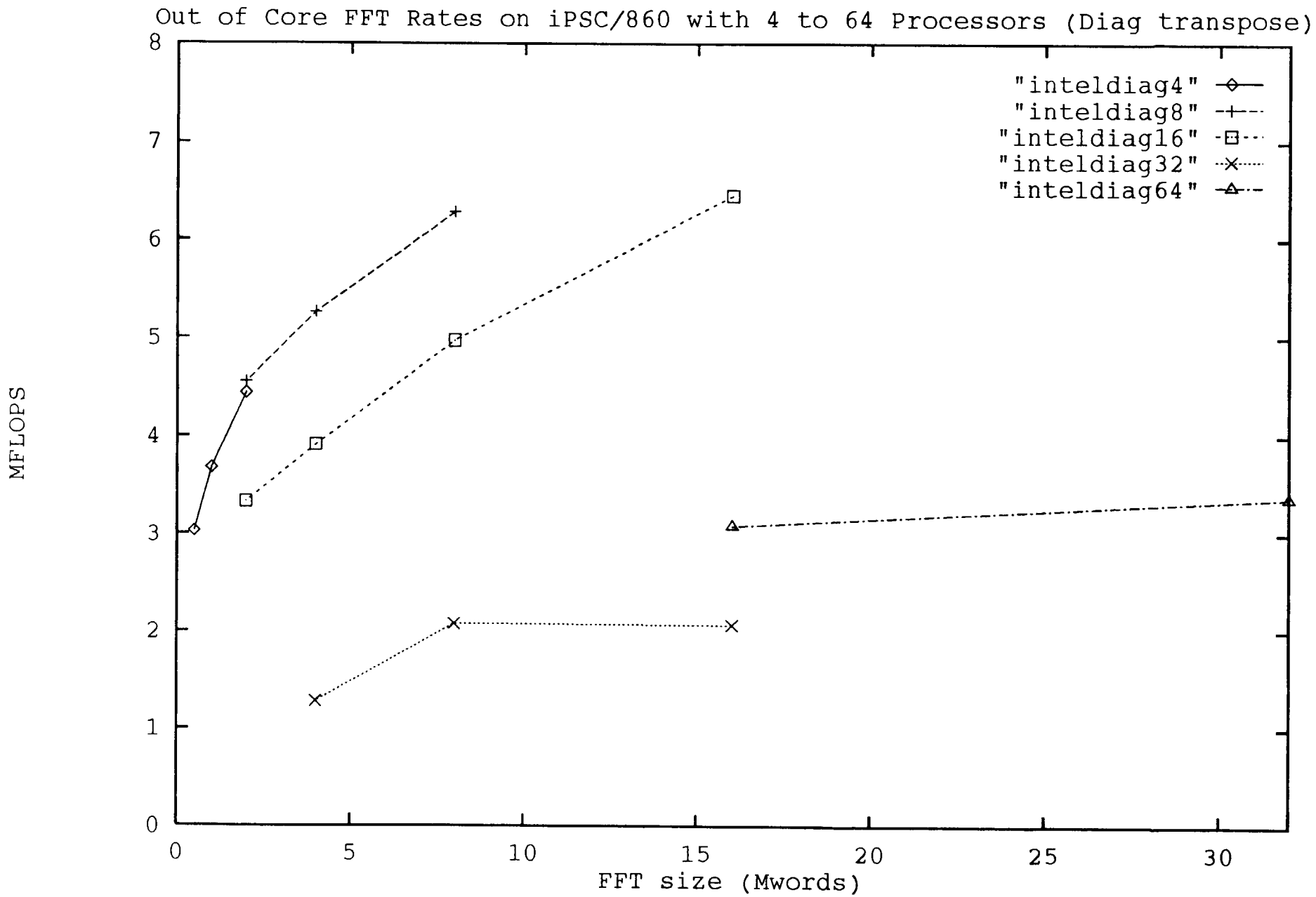
Again, the diagonal method was not feasible to implement on the CM5. On the Intel, for the same reason the row/col method was not pessimal, the diagonal method was not optimal. Repeated measurements did show a mild superiority of the diagonal method over the row/col method.

Graph 2, for the diagonal method, shows a similar performance profile as seen in graph 1, for the row/col transpose, except that the performance numbers are generally 1 MFLOP faster. This represents a significant performance improvement over the row/col method (20 percent in the case of 5MFLOP row/col configurations). However, this improvement does not reflect the asymptotic superiority suggested in [5].

Problem size and number of processors are much more significant than the particular choice of data movement algorithm. For example, consider the question: What is the most significant performance improvement option when solving a 16 Mword FFT with 32 processors using the row/col method? If we reduce the number of processors to 16, then we see a gain in performance from 1 MFLOP to 5 MFLOP. If instead we switch to the diagonal method, we only gain from 1 MFLOP to 2 MFLOP.

From this graph we also see that the iPSC/860 exhibits poor processor scaling — improving up to 16 processors then declining in the same way as shown in [8].

Graph 2

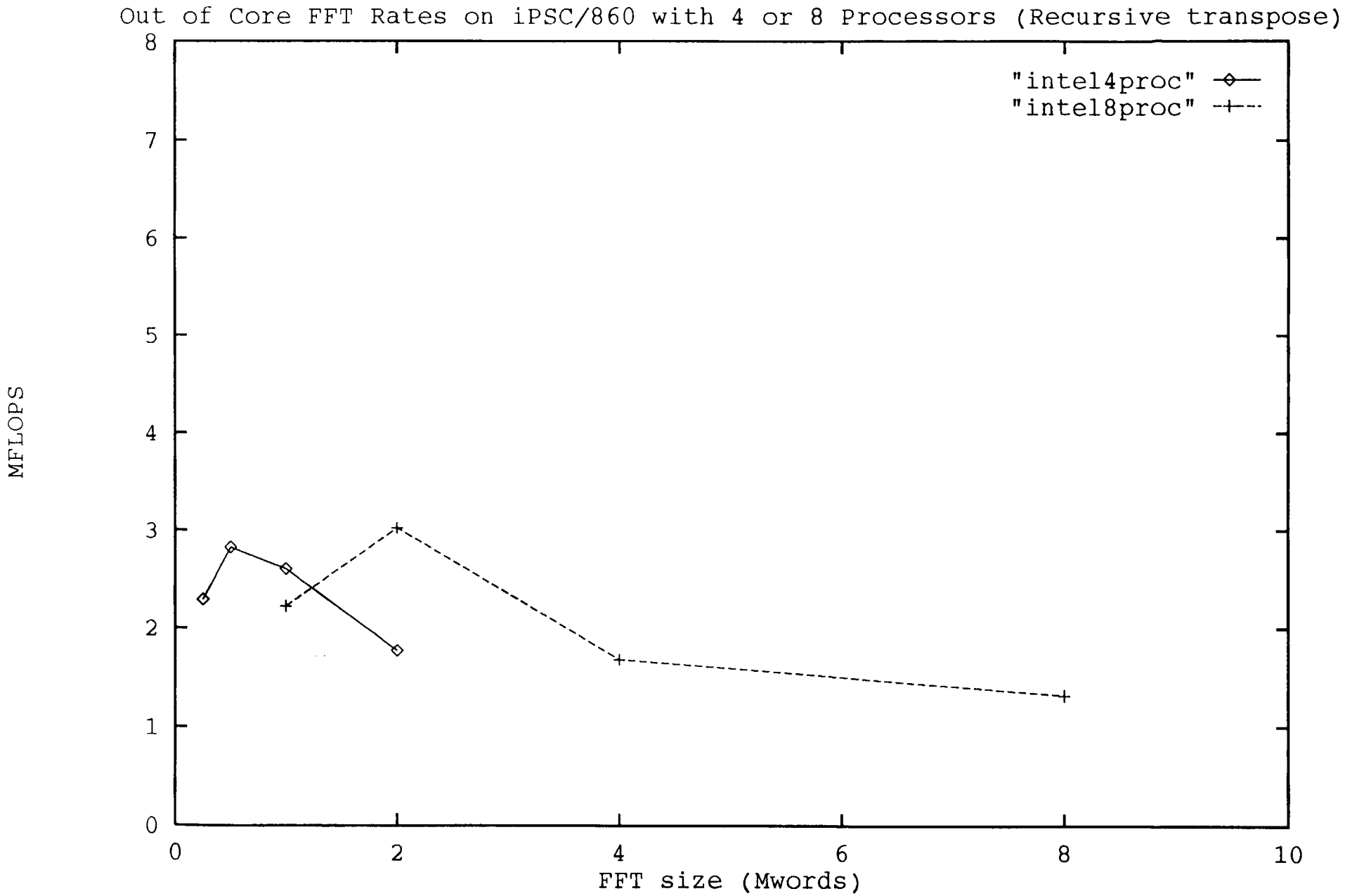

Out of Core FFT Rates on iPSC/860 with 4 to 64 Processors (Diag transpose)
MFLOPS
FFT size (Mwords)
"inteldiag4"
"inteldiag8"
"inteldiag16"
"inteldiag32"
"inteldiag64"
0
1
2
3
4
5
6
7
8
0
5
10
15
20
25
30

3.1.3 Recursive Method

The CM5 performed best with the recursive transpose method. In fact, this was the only feasible choice for the CM5 because parallelism was not possible for the CM5 for either of the other algorithms. The iPSC/860 performed worst on the recursive transpose method. Since problem sizes that qualify as out of core increase with the number of processors, and performance drops rapidly with problem size, no out of core solutions using this approach were attempted for more than 8 processors on the iPSC/860.

Graph 3 shows the performance of the iPSC/860 in solving out of core FFTs for sizes varying from .25 Mwords to as many as 8 Mwords. The decrease in MFLOP rate as problem size increases shows that the recursive transpose algorithm as implemented on the iPSC/860 does not scale well, whether on 4 processors or on 8.

Graph 3

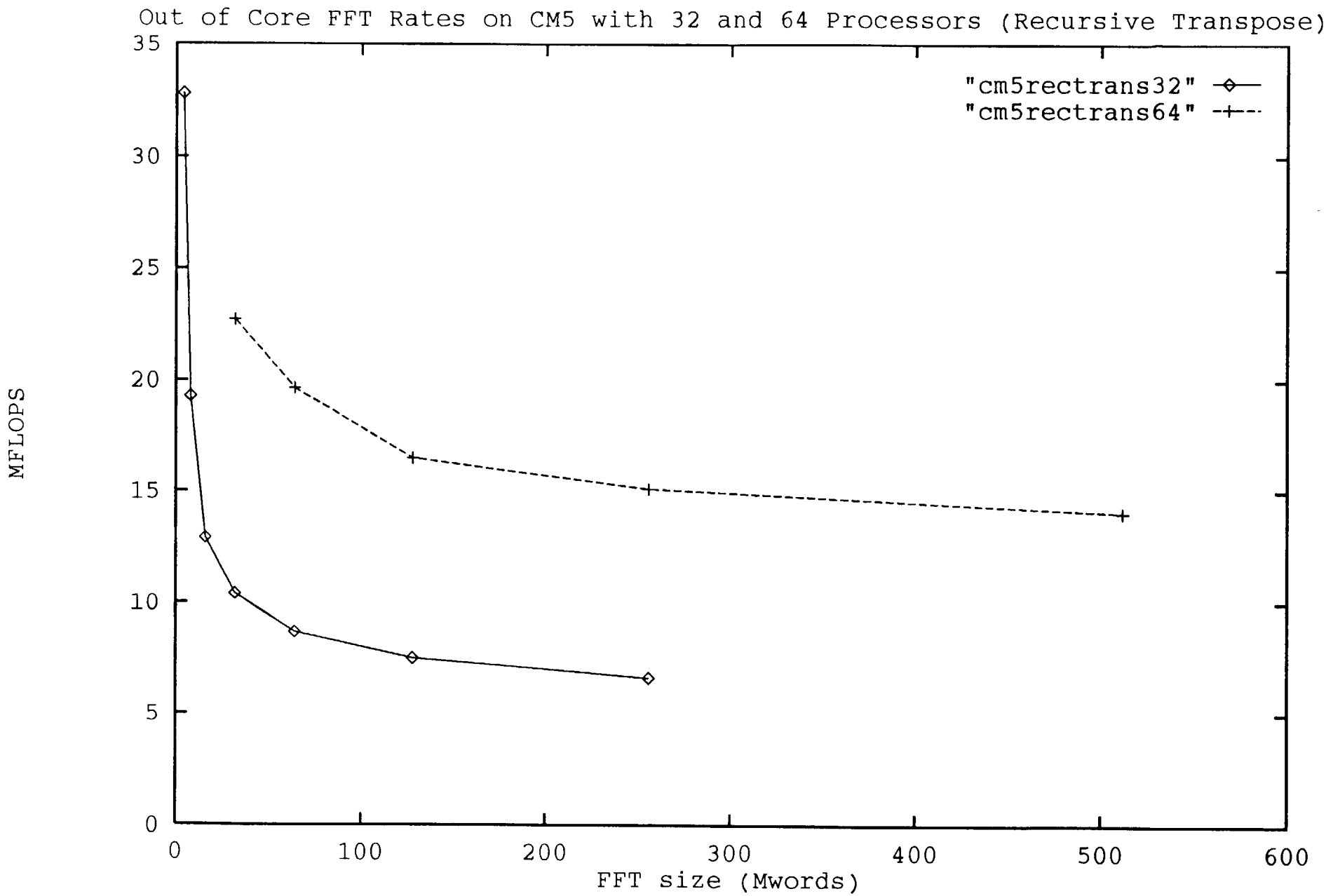

Out of Core FFT Rates on iPSC/860 with 4 or 8 Processors (Recursive transpose)
"intel4proc"
"intel8proc"
MFLOPS
FFT size (Mwords)
0
1
2
3
4
5
6
7
8
0
2
4
6
8
10

Graph 4 shows the out of core FFT performance for the CM5 for problem sizes ranging from 4 Mwords to as many as 512 Mwords. We see that as problem size increases, performance drops. Beside this basic observation, there are two key features of the performance illustrated by graph 4.

The first feature of interest is the rapid *initial* reduction in performance. This corresponds to the logarithmic growth of the number of recursions this algorithm makes. At first, the logarithmic growth is fairly rapid, resulting in a rapid reduction in performance. Later, the logarithm grows proportionately less rapidly, and so the performance declines more slowly. A similar but less striking effect is also seen on graph 3.

The second issue of interest is the scale of graph 4 compared to graphs 1, 2 and 3. The graphs suggest that the iPSC/860 scales better than the CM5 — but the entire set of iPSC/860 graphs fits in a small square in the lower left hand corner of the CM5 graph, bounded above at 7 MFLOPS and to the right at 32 Mwords. The CM5 SDA system garners superior absolute performance for all problem sizes.

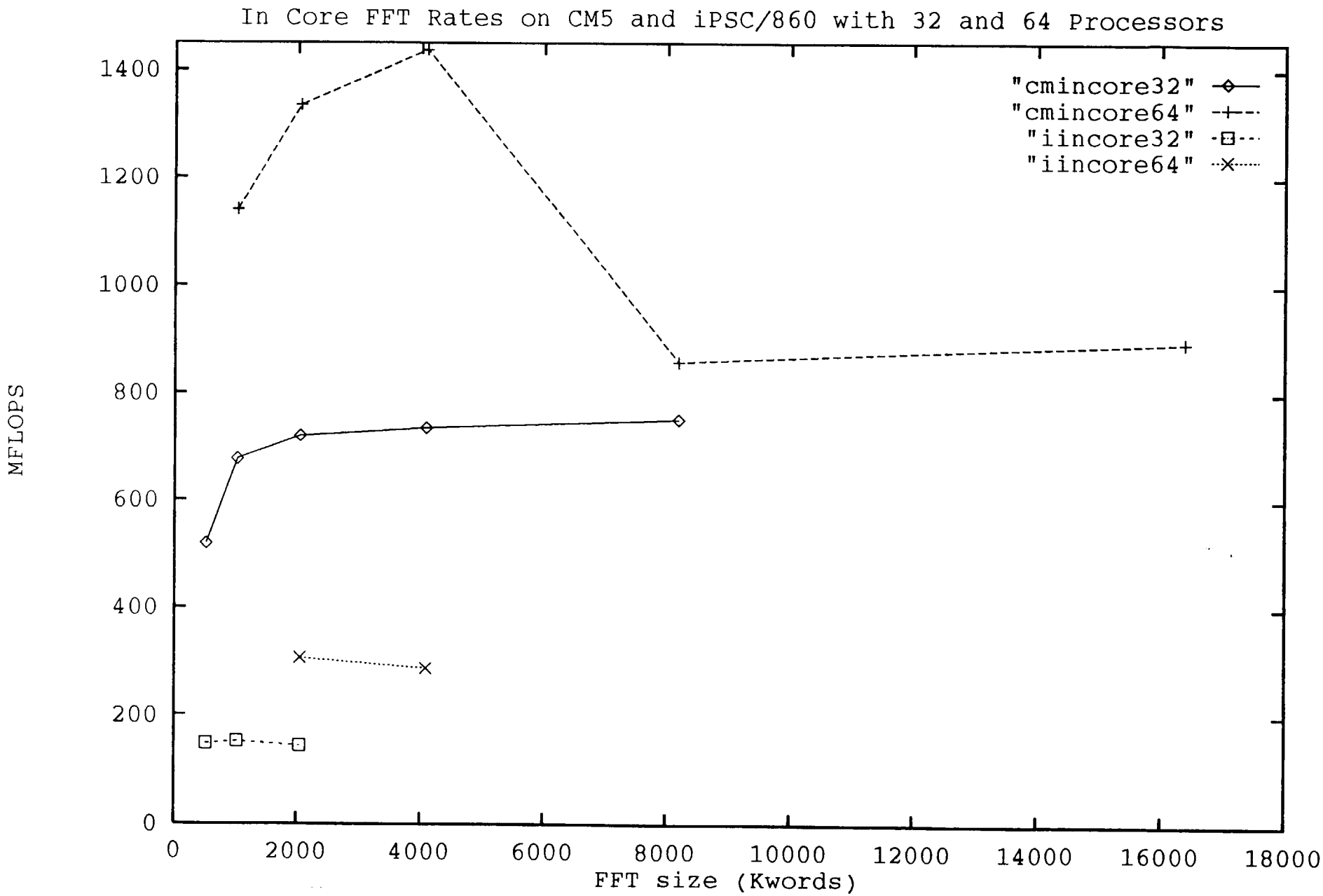
Graph 4


Out of Core FFT Rates on CM5 with 32 and 64 Processors (Recursive Transpose)
"cm5rectrans32"
"cm5rectrans64"
MFLOPS
FFT size (Mwords)
0
5
10
15
20
25
30
35
0
100
200
300
400
500
600

3.2 In Core FFT Performance

In order to better understand the significance of our results, it is useful to examine the in core performance of the iPSC/860 and CM5 for the FFT. The following graph shows the performance of the CM5 and the iPSC/860 for a variety of FFT problem sizes. The CM5 was able to perform much larger in core FFTs than the iPSC/860. Clearly, the in core rates are overwhelmingly faster than the out of core rates. We also see that while the FFTs are in core, they scale well with problem size and number of processors (although there is a curious drop in performance for the CM5 with 64 processors between a 4Mword and 8Mword problem size.). Finally, we see that the CM5 outperforms the iPSC/860 by roughly a factor of 5.

Graph 5


In Core FFT Rates on CM5 and iPSC/860 with 32 and 64 Processors
MFLOPS
FFT size (Kwords)
"cmincore32"
"cmincore64"
"iincore32"
"iincore64"
0
200
400
600
800
1000
1200
1400
0
2000
4000
6000
8000
10000
12000
14000
16000
18000

4 Conclusion

For out of core FFT algorithms, the CM5 scales poorly with problem size, but scales well with the number of processors. The iPSC/860 scales poorly with the number of processors, but well with the problem size. We find some of the recommendations for algorithms and architectural capabilities given by the literature [5] to be valuable. In particular, the need for independent read head access, and its relationship to the scalability of problem solutions is confirmed by the good scaling properties exhibited by the row/col and diagonal methods on the iPSC/860. We also confirm that in core computational capacity is insignificant compared to I/O performance and flexibility when considering out of core FFT solution speed.

We conclude then that one key issue in parallel I/O architecture design is the tradeoff between an architecture without independent access and high raw performance and a flexible architecture with independent access, but lower raw performance. In this study, the superior raw performance of the CM5 SDA outweighed its poor scaling.

5 Future Work

It is not clear that architectures with independent access necessarily have lower raw performance than those without independent access. The Intel Paragon with PFS should be evaluated using the same methods applied here to determine the scalability of the more flexible design entailed in CFS. Also, the restrictvol feature of CFS and PFS could provide for higher raw performance. The restricvol command directs to which subset of disks a given file is stored. This capability should be explored.

6 Acknowledgements

Thanks to Bill Nitzberg and Russell Carter for reviewing this paper.

References

[1] Aggarwal, A. Vitter, J 'The Input/Output Complexity of Sorting and

Related Problems' Communications of the ACM, 31(9), September 1988, 1116–1127.

[2] Bailey, D. H. 'FFTs in External or Hierarchical Memory', *Proceedings of Supercomputing*, 1989.

[3] Bailey, Barton, Lasinski, Simon *The NAS Parallel Benchmarks* NAS Report RNR-91-002, 22 August, 1991.

[4] Brigham, E. O. *The Fast Fourier Transform*, Prentice Hall, Englewood Cliffs, NJ, 1974

[5] Cormen, T, Kotz, D. "Integrating Theory and Practice in Parallel File Systems." *DAGS '93*, Hanover NH, pages 64-74

[6] *iPSC/860 Programmers Reference Manual* Intel Corporation Document.

[7] *NAS User Guide Version 6.0* Vol. 4, Numerical Aerodynamic Simulation Facility, NASA Ames research Center, Moffett Field Ca.

[8] Nitzberg, B. *Performance of the iPSC/860 Concurrent File System* NAS Technical Report RND-92-020, December 1992.

[9] Patterson, D. A., Hennessey J. L. *Computer Architecture: a Quantitative Approach* Morgan Kaufmann, San Mateo Ca. 1990

[10] Scott, D. S. 'Parallel I/O and Solving Out of Core Systems of Linear Equations' *DAGS '93*, Hanover NH, pages 123–130

[11] CM-5 I/O System Programming Guide, Thinking Machines Corporation Document December 21, 1992.

[12] Vitter, J. Shriver E. *Optimal Disk I/O with Parallel Block Transfer* Proceedings of the 22nd Annual ACM Symposium on Theory of computing. May, 1990 pages 159-169.

RND TECHNICAL REPORT

Title: Out of Core FFTs in a parallel application environment.

Author(s): Christopher Lee Kuszmaul

Reviewers:

"I have carefully and thoroughly reviewed this technical report. I have worked with the author(s) to ensure clarity of presentation and technical accuracy. I take personal responsibility for the quality of this document."

Two reviewers must sign.

Signed: Bill Nitzberg

Name: Bill Nitzberg

Signed: Russell Carter

Name: Russell Carter

After approval, assign RND TR number.

Branch Chief:

Approved: for BTB

Date: 8-10-93

TR Number: RND-93-013

Important: Put this form as the last page in the published Tech Report.